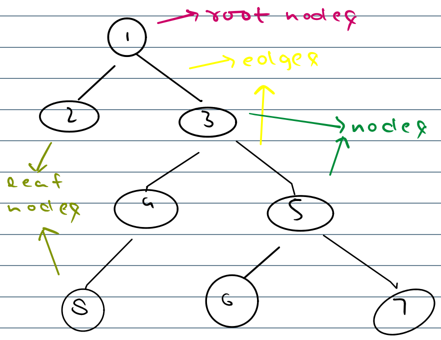
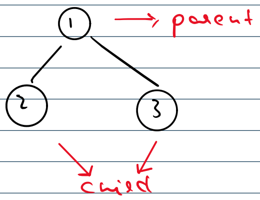
Trees

It is a non-linear data structure and it represents data in hierarchical form.

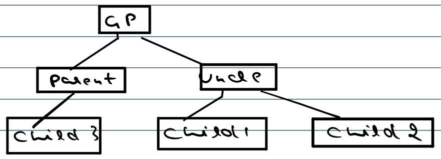




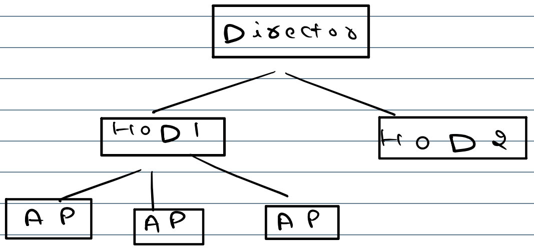
1. Nodes(N) → Edges(N-1)
2. single parent except node
3. any no. of children
4. no cycles in tree.

Real world analogies

1. Family Tree

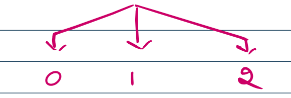


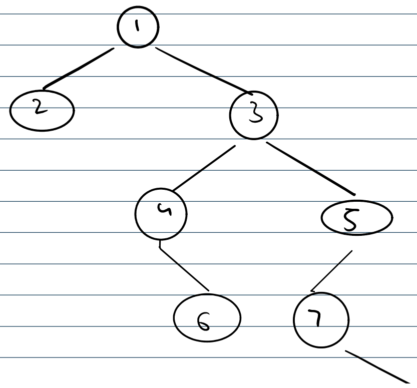
2. organisation Tree

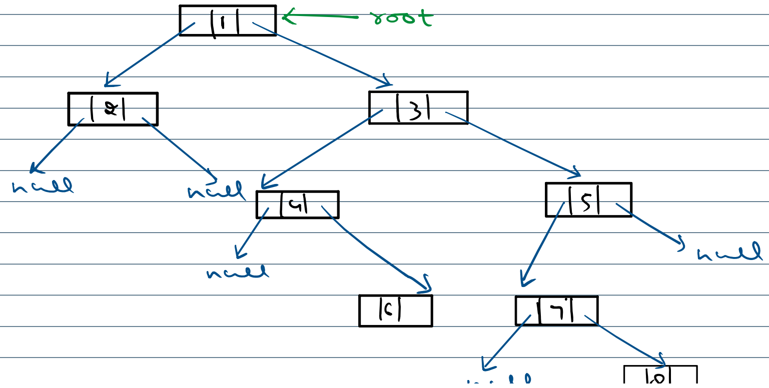


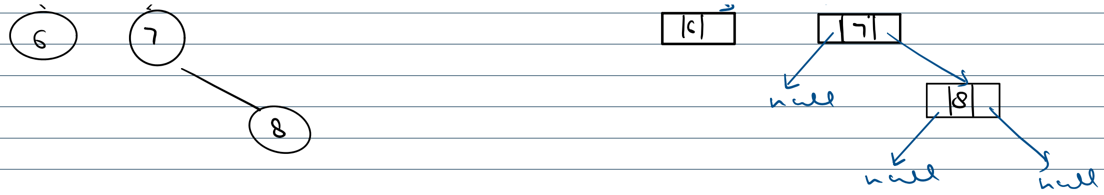
Binary Tree

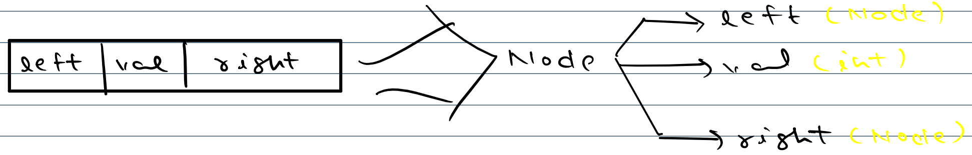
where each node has at most two children (left & right child)









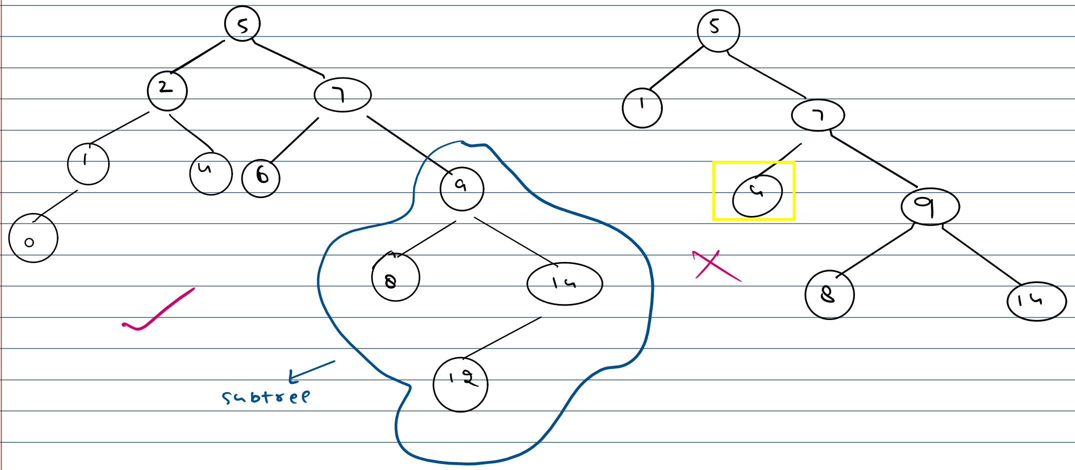


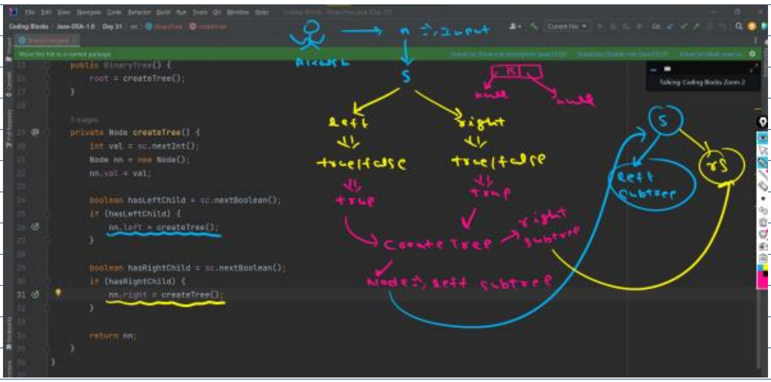
Binary Search Tree

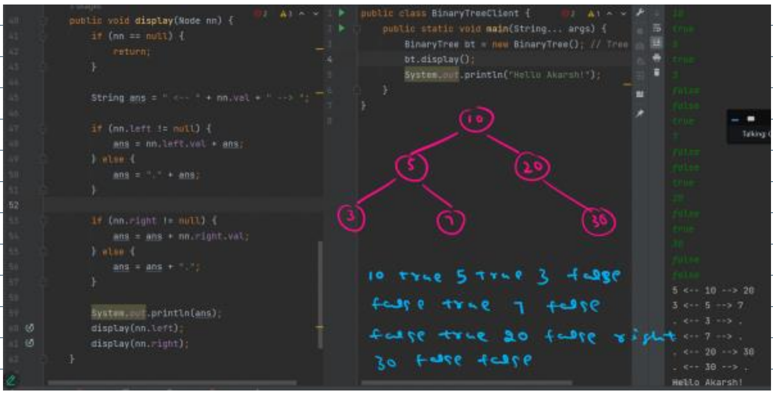
It is a BT that follows:

ordering property → left child :) smaller than parent
→ right child :) greater than parent

Example

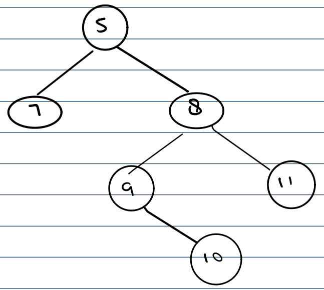



```
public BinaryTree() {
    root = createTree();
}

private Node createTree() {
    int val = sc.nextInt();
    Node nn = new Node();
    nn.val = val;

    boolean hasLeftChild = sc.nextBoolean();
    if (hasLeftChild) {
        nn.left = createTree();
    }

    boolean hasRightChild = sc.nextBoolean();
    if (hasRightChild) {
        nn.right = createTree();
    }

    return nn;
}
}
```


```
public void display(Node nn) {
    if (nn == null) {
        return;
    }

    String ans = " <-- " + nn.val + " --> ";

    if (nn.left != null) {
        ans = nn.left.val + ans;
    } else {
        ans = "." + ans;
    }

    if (nn.right != null) {
        ans = ans + nn.right.val;
    } else {
        ans = ans + ".";
    }

    System.out.println(ans);
    display(nn.left);
    display(nn.right);
}
```

```
public class BinaryTreeClient {
    public static void main(String... args) {
        BinaryTree bt = new BinaryTree(); // Tree
        bt.display();
        System.out.println("Hello Akarsh!");
    }
}
```

```
10
true
5
true
3
false
false
true
7
false
false
true
20
false
true
30
false
false
5 <-- 10 --> 20
3 <-- 5 --> 7
. <-- 3 --> .
. <-- 7 --> .
. <-- 20 --> 30
. <-- 30 --> .
Hello Akarsh!
```



5

true

7

false

false

true

8

true

9

false

true

10

false

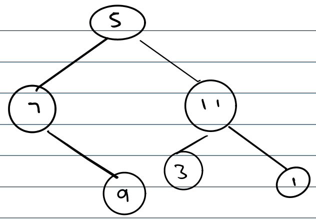
false

true

11

false

false



5

true

7

false

true

9

false

false

true

11

true

3

false

false

true

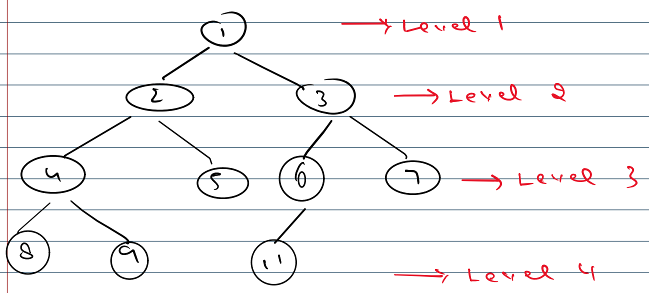
1

false

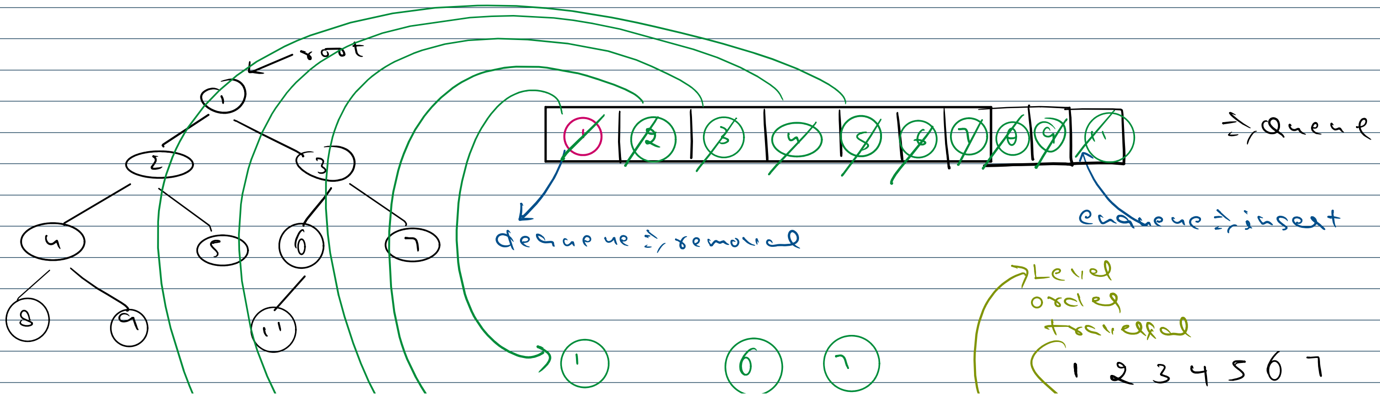
false

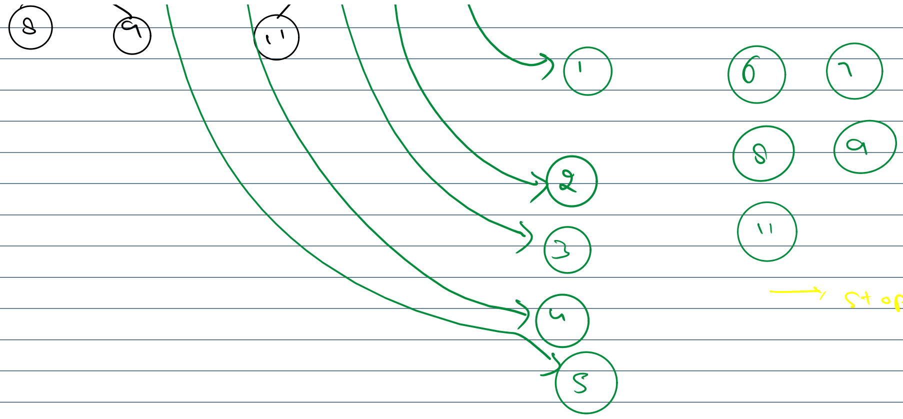
Traversals

Level order traversal



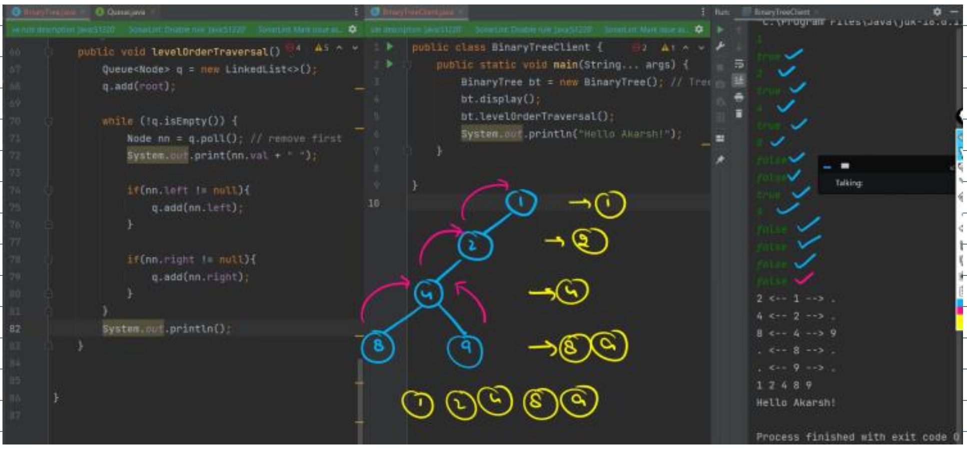
1, 2, 3, 4, 5, 6, 7, 8, 9, 11





order traversal
1 2 3 4 5 6 7
8 9 11

→ stops → when Q is empty



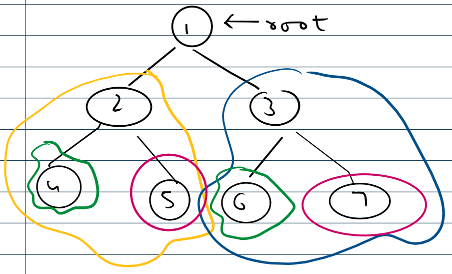
```java
    public void levelOrderTraversal() {
        Queue<Node> q = new LinkedList<>();
        q.add(root);

        while (!q.isEmpty()) {
            Node nn = q.poll(); // remove first
            System.out.print(nn.val + " ");

            if(nn.left != null){
                q.add(nn.left);
            }

            if(nn.right != null){
                q.add(nn.right);
            }
        }
        System.out.println();
    }


}
```

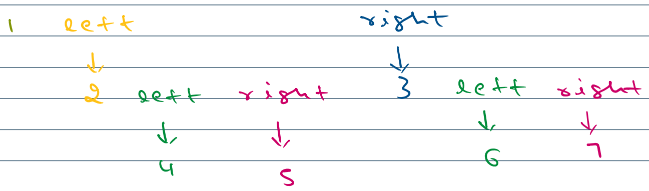
```java
public class BinaryTreeClient {
    public static void main(String... args) {
        BinaryTree bt = new BinaryTree(); // Tree
        bt.display();
        bt.levelOrderTraversal();
        System.out.println("Hello Akarsh!");
    }

}
```

```
C:\Program Files\Java\jdk-18.0.1
1
true
2
true
4
true
8
false
false
true
9
false
false
false
false
2 <-- 1 --> .
4 <-- 2 --> .
8 <-- 4 --> 9
. <-- 8 --> .
. <-- 9 --> .
1 2 4 8 9
Hello Akarsh!

Process finished with exit code 0
```

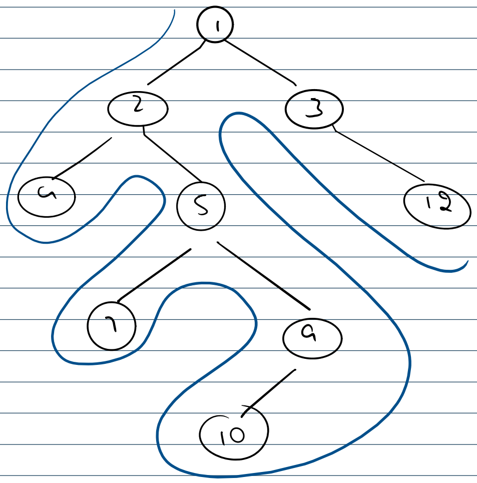
Traversal
→ PreOrder ⇒ val → left → right
→ PostOrder ⇒ left → right → val
→ Inorder ⇒ left → val → right

PreOrder Traversal



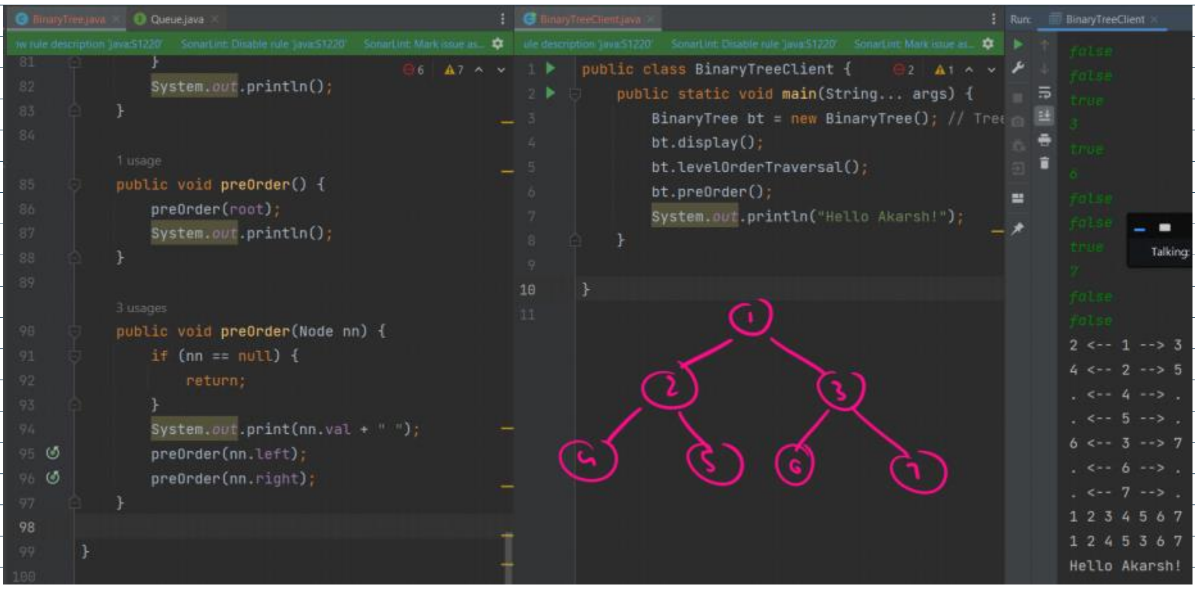


1 2 4 5 3 6 7



1 2 4 5 7 9 10 3 12

```
1
true
2
true
4
false
false
true
5
false
false
true
3
true
6
false
false
true
7
false
false
```

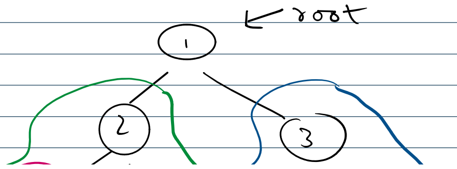


```java
        }
        System.out.println();
    }

    1 usage
    public void preOrder() {
        preOrder(root);
        System.out.println();
    }

    3 usages
    public void preOrder(Node nn) {
        if (nn == null) {
            return;
        }
        System.out.print(nn.val + " ");
        preOrder(nn.left);
        preOrder(nn.right);
    }

}
```

```java
public class BinaryTreeClient {
    public static void main(String... args) {
        BinaryTree bt = new BinaryTree(); // Tree
        bt.display();
        bt.levelOrderTraversal();
        bt.preOrder();
        System.out.println("Hello Akarsh!");
    }

}
```

```
false
false
true
3
true
6
false
false
true
7
false
false
2 <-- 1 --> 3
4 <-- 2 --> 5
. <-- 4 --> .
. <-- 5 --> .
6 <-- 3 --> 7
. <-- 6 --> .
. <-- 7 --> .
1 2 3 4 5 6 7
1 2 4 5 3 6 7
Hello Akarsh!
```

Post order Traversal

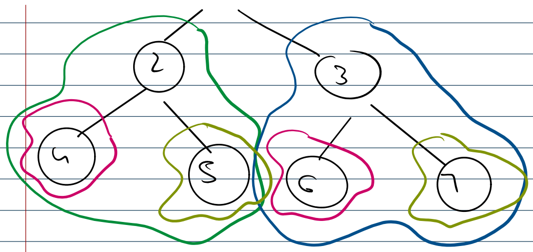


left   right   1

↓   ↓

left   right   2   left   right   3

↓   ↓   ↓   ↓

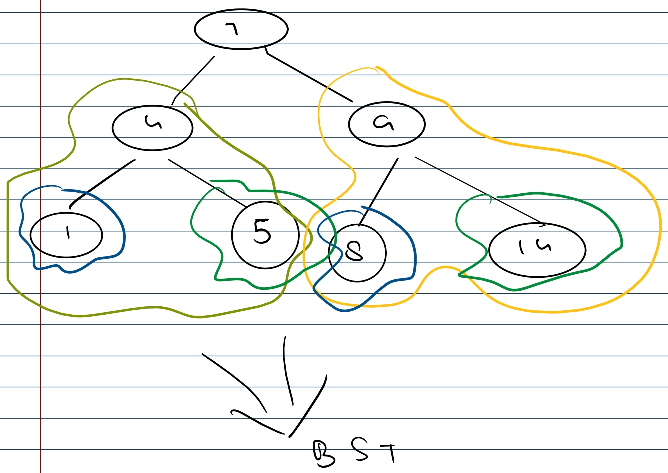


left right 2 left right 3

4 5 6 7

4 5 2 6 7 3 1

Inorder traversal



left 7 right

left 4 right left 9 right

1 5 8 14

1 4 5 7 8 9 14

always sorted for BST